On the Hardness of Almost–Sure Termination*

Benjamin Lucien Kaminski and Joost-Pieter Katoen

Software Modeling and Verification Group
RWTH Aachen University
{benjamin.kaminski,katoen}@cs.rwth-aachen.de

Abstract. This paper considers the computational hardness of computing expected outcomes and deciding (universal) (positive) almost–sure termination of probabilistic programs. It is shown that computing lower and upper bounds of expected outcomes is Σ^0_1– and Σ^0_2–complete, respectively. Deciding (universal) almost–sure termination as well as deciding whether the expected outcome of a program equals a given rational value is shown to be Π^0_2–complete. Finally, it is shown that deciding (universal) positive almost–sure termination is Σ^0_2–complete (Π^0_3–complete).

Keywords: probabilistic programs · expected outcomes · almost–sure termination · positive almost–sure termination · computational hardness

1 Introduction

Probabilistic programs [1] are imperative programs with the ability to toss a (possibly) biased coin and proceed their execution depending on the outcome of the coin toss. They are used in randomized algorithms, in security to describe cryptographic constructions (such as randomized encryption) and security experiments [2], and in machine learning to describe distribution functions that are analyzed using Bayesian inference [3]. Probabilistic programs are typically just a small number of lines, but hard to understand and analyze, let alone algorithmically. This paper considers the computational hardness of two main analysis problems (and variations thereof) for probabilistic programs:

1. Computing expected outcomes: Is the expected outcome of a program variable smaller than, equal to, or larger than a given rational number?
2. Deciding [universal] (positive) almost–sure termination: Does a program terminate [on all inputs] with probability one (within an expected finite number of computation steps)?

The first analysis problem is related to determining weakest pre–expectations of probabilistic programs [4,5]. Almost–sure termination is an active field of research [6]. A lot of work has been done towards automated reasoning for almost–sure termination. For instance, [7] gives an overview of some particularly interesting examples of probabilistic logical programs and the according

* This research is funded by the Excellence Initiative of the German federal and state governments and by the EU FP7 MEALS project.

intuition for proving almost–sure termination. Arons *et al.* [8] reduce almost–sure termination to termination of non–deterministic programs by means of a planner. This idea has been further exploited and refined into a pattern–based approach with prototypical tool support [9].

Despite the existence of several (sometimes automated) approaches to tackle almost–sure termination, most authors claim that it must intuitively be harder than the termination problem for ordinary programs. To mention a few, Morgan [10] remarks that while partial correctness for small–scale examples is not harder to prove than for ordinary programs, the case for total correctness of a probabilistic loop must be harder to analyze. Esparza *et al.* [9] claim that almost–sure termination must be harder to decide than ordinary termination since for the latter a topological argument suffices while for the former arithmetical reasoning is needed. The computational hardness of almost–sure termination has however received scant attention. As a notable exception, [11] establishes that deciding almost–sure termination of certain concurrent probabilistic programs is in Π^0_2.

In this paper, we give precise classifications of the level of arithmetical reasoning that is needed to decide the aforementioned analysis problems by establishing the following results: We first show that computing lower bounds on the expected outcome of a program variable v after executing a probabilistic program P on a given input η is Σ^0_1–complete and therefore arbitrarily close approximations from below are computable. Computing upper bounds, on the other hand, is shown to be Σ^0_2–complete, thus arbitrarily close approximations from above are not computable in general. Deciding whether an expected outcome equals some rational is shown to be Π^0_2–complete.

For the second analysis problem—almost–sure termination—we obtain that deciding almost–sure termination of probabilistic program P on a given input η is Π^0_2–complete. While for ordinary programs we have a complexity leap when moving from the non–universal to the universal halting problem, we establish that *this is not the case for probabilistic programs*: Deciding universal a.s termination turns out to be Π^0_2–complete too. The case for *positive* almost–sure termination is different however: While deciding (non–universal) positive almost–sure termination is Σ^0_2–complete, we show that universal positive almost–sure termination is Π^0_3–complete.

2 Preliminaries

As indicated, our hardness results will be stated in terms of levels in the arithmetical hierarchy—a concept we briefly recall:

Definition 1 (Arithmetical Hierarchy [12,13]). *For every $n \in \mathbb{N}$, the* ***class*** $\boldsymbol{\Sigma^0_n}$ *is defined as* $\Sigma^0_n = \big\{\mathcal{A} \;\big|\; \mathcal{A} = \big\{x \;\big|\; \exists y_1\, \forall y_2\, \exists y_3 \cdots \exists/\forall y_n \colon\ (x, y_1, y_2, y_3, \ldots, y_n) \in \mathcal{R}\big\},\ \mathcal{R} \text{ is a decidable relation}\big\}$, *the* ***class*** $\boldsymbol{\Pi^0_n}$ *is defined as* $\Pi^0_n = \big\{\mathcal{A} \;\big|\; \mathcal{A} = \big\{x \;\big|\; \forall y_1\, \exists y_2\, \forall y_3 \cdots \exists/\forall y_n \colon\ (x, y_1, y_2, y_3, \ldots, y_n) \in \mathcal{R}\big\},\ \mathcal{R} \text{ is a decidable relation}\big\}$ *and the* ***class*** $\boldsymbol{\Delta^0_n}$ *is defined as* $\Delta^0_n = \Sigma^0_n \cap \Pi^0_n$. *Note that we require that the*

values of the variables are drawn from a recursive domain. Multiple consecutive quantifiers of the same type *can be contracted to* one *quantifier of that type, so the number n really refers to the number of necessary* quantifier alternations *rather than to the number of quantifiers used. A set* $\mathcal{A}$ *is called* ***arithmetical****, iff* $\mathcal{A} \in \Gamma_n^0$*, for* $\Gamma \in \{\Sigma, \Pi, \Delta\}$ *and* $n \in \mathbb{N}$*. The arithmetical sets form a strict hierarchy, i.e.* $\Delta_n^0 \subset \Gamma_n^0 \subset \Delta_{n+1}$ *and* $\Sigma_n^0 \neq \Pi_n^0$ *holds for* $\Gamma \in \{\Sigma, \Pi\}$ *and* $n \geq 1$*. Furthermore, note that* $\Sigma_0^0 = \Pi_0^0 = \Delta_0^0 = \Delta_1^0$ *is exactly the class of the decidable sets and* Σ_1^0 *is exactly the class of the recursively enumerable sets.*

Next, we recall the concept of many–one reducibility and completeness:

Definition 2 (Many–One Reducibility and Completeness [13,14,15]). *Let* $\mathcal{A}, \mathcal{B}$ *be arithmetical sets and let* X *be some appropriate universe such that* $\mathcal{A}, \mathcal{B} \subseteq X$*.* $\mathcal{A}$ *is called* ***many–one–reducible*** *to* $\mathcal{B}$*, denoted* $\boldsymbol{\mathcal{A} \leq_{\mathrm{m}} \mathcal{B}}$*, iff there exists a computable function* $f\colon X \to X$*, such that* $\forall\, x \in X\colon \big(x \in \mathcal{A} \iff f(x) \in \mathcal{B}\big)$*. If* f *is a function such that* f *many–one reduces* $\mathcal{A}$ *to* $\mathcal{B}$*, we denote this by* $\boldsymbol{f\colon \mathcal{A} \leq_{\mathrm{m}} \mathcal{B}}$*. Note that* $\leq_{\mathrm{m}}$ *is transitive.*

$\mathcal{A}$ *is called* $\boldsymbol{\Gamma_n^0}$***–complete****, for* $\Gamma \in \{\Sigma, \Pi, \Delta\}$*, iff both* $\mathcal{A} \in \Gamma_n^0$ *and* $\mathcal{A}$ *is* $\boldsymbol{\Gamma_n^0}$***–hard****, meaning* $\mathcal{C} \leq_{\mathrm{m}} \mathcal{A}$*, for any set* $\mathcal{C} \in \Gamma_n^0$*. Note that if* $\mathcal{A}$ *is* Γ_n^0*–complete and* $\mathcal{A} \leq_{\mathrm{m}} \mathcal{B}$*, then* $\mathcal{B}$ *is necessarily* Γ_n^0*–hard. Furthermore, note that if* $\mathcal{A}$ *is* Σ_n^0*–complete, then* $\mathcal{A} \in \Sigma_n^0 \backslash \Pi_n^0$*. Analogously if* $\mathcal{A}$ *is* Π_n^0*–complete, then* $\mathcal{A} \in \Pi_n^0 \backslash \Sigma_n^0$*.*

3 Probabilistic Programs

In order to speak about probabilistic programs and the computations performed by such programs, we briefly introduce the syntax and semantics we use:

Definition 3 (Syntax). *Let* Var *be the set of program variables. The* ***set*** $\mathbf{Prog}$ ***of probabilistic programs*** *adheres to the following grammar:*

$$\mathsf{Prog} \quad \longrightarrow \quad v \mathtt{:=} e \mid \mathsf{Prog}\,\mathtt{;}\,\mathsf{Prog} \mid \{\mathsf{Prog}\}\,[p]\,\{\mathsf{Prog}\} \mid \mathtt{WHILE}\,(b)\,\{\mathsf{Prog}\}\ ,$$

where $v \in \mathsf{Var}$*,* e *is an arithmetical expression over* Var*,* $p \in [0, 1] \subseteq \mathbb{Q}$*, and* b *is a Boolean expression over arithmetic expressions over* Var*. We call the set of programs that* do not *contain any probabilistic choices the* ***set of ordinary programs*** *and denote this set by* $\mathbf{ordProg}$*.*

The presented syntax is the one of the fully probabilistic[1] fragment of the probabilistic guarded command language (pGCL) originally due to McIver and Morgan [4]. We omitted `skip`–, `abort`–, and `if`–statements, as those are syntactic sugar. While assignment, concatenation, and the while–loop are standard programming constructs, $\{P_1\}\,[p]\,\{P_2\}$ denotes a probabilistic choice between programs P_1 (with probability p) and P_2 (with probability $1 - p$). An operational semantics for pGCL programs is given below:

[1] Fully probabilistic programs may contain probabilistic but no non–deterministic choices.

Definition 4 (Semantics). *Let the set of variable valuations be denoted by* $\mathbb{V} = \{\eta \mid \eta\colon \mathsf{Var} \to \mathbb{Q}^+\}$*, let the set of program states be denoted by* $\mathbb{S} = \big(\mathsf{Prog} \cup \{\downarrow\}\big) \times \mathbb{V} \times I \times \{L, R\}^*$*, for* $I = [0, 1] \cap \mathbb{Q}^+$*, let* $[\![e]\!]_\eta$ *be the evaluation of the arithmetical expression* e *in the variable valuation* η*, and analogously let* $[\![b]\!]_\eta$ *be the evaluation of the Boolean expression* b*. Then the* ***semantics of probabilistic programs*** *is given by the smallest relation* $\vdash \subseteq \mathbb{S} \times \mathbb{S}$ *which satisfies the following inference rules:*

$$\text{(assign)}\ \frac{}{\langle v\ \texttt{:=}\ e,\, \eta,\, a,\, \theta\rangle\ \vdash\ \langle\downarrow,\, \eta[v \mapsto \max\{[\![e]\!]_\eta,\, 0\}],\, a,\, \theta\rangle}$$

$$\text{(concat1)}\ \frac{\langle P_1,\, \eta,\, a,\, \theta\rangle\ \vdash\ \langle P_1',\, \eta',\, a',\, \theta'\rangle}{\langle P_1\,\texttt{;}\ P_2,\, \eta,\, a,\, \theta\rangle\ \vdash\ \langle P_1'\texttt{;}\ P_2,\, \eta',\, a',\, \theta'\rangle}$$

$$\text{(concat2)}\ \frac{}{\langle\downarrow\texttt{;}\ P_2,\, \eta,\, a,\, \theta\rangle\ \vdash\ \langle P_2,\, \eta,\, a,\, \theta\rangle}$$

$$\text{(prob1)}\ \frac{}{\langle\{P_1\}\ [p]\ \{P_2\},\, \eta,\, a,\, \theta\rangle\ \vdash\ \langle P_1,\, \eta,\, a \cdot p,\, \theta \cdot L\rangle}$$

$$\text{(prob2)}\ \frac{}{\langle\{P_1\}\ [p]\ \{P_2\},\, \eta,\, a,\, \theta\rangle\ \vdash\ \langle P_2,\, \eta,\, a \cdot (1-p),\, \theta \cdot R\rangle}$$

$$\text{(while1)}\ \frac{[\![b]\!]_\eta = \text{True}}{\langle\texttt{WHILE}\ (b)\ \{P\},\, \eta,\, a,\, \theta\rangle\ \vdash\ \langle P\texttt{;}\ \texttt{WHILE}\ (b)\ \{P\},\, \eta,\, a,\, \theta\rangle}$$

$$\text{(while2)}\ \frac{[\![b]\!]_\eta = \text{False}}{\langle\texttt{WHILE}\ (b)\ \{P\},\, \eta,\, a,\, \theta\rangle\ \vdash\ \langle\downarrow,\, \eta,\, a,\, \theta\rangle}$$

We use $\sigma \vdash^k \tau$ *in the usual sense.*

The semantics is mostly straightforward except for two features: in addition to the program that is to be executed next and the current variable valuation, each state also stores a sequence θ that encodes which probabilistic choices were made in the past (Left or Right) as well as the probability a that those choices were made. The graph that is spanned by the $\vdash$–relation can be seen as an unfolding of the Markov decision process semantics for pGCL provided by Gretz *et al.* [5] when restricting oneself to fully probabilistic programs.

4 Expected Outcomes and Termination Probabilities

In this section we formally define the notion of an expected outcome as well the notion of (universal) (positive) almost–sure termination. We start by investigating how state successors can be computed.

It is a well–known result due to Kleene that for any ordinary program P and a state σ the k-th successor of σ with respect to $\vdash$ is unique and computable. If, however, P is a probabilistic program containing probabilistic choices, the k-th successor of a state need not be unique, because at various points of the execution

the program must choose a left or a right branch with some probability. However, if we resolve those choices by providing a sequence of symbols w over the alphabet $\{L, R\}$ that encodes for all probabilistic choices which occur whether the *L*eft or the *R*ight branch shall be chosen at a branching point, we can construct a computable function that computes a unique k-th successor. Notice that for this purpose a sequence of finite length is sufficient. We obtain the following:

Proposition 1 (The State Successor Function). *Let* $\mathbb{S}_\bot = \mathbb{S} \cup \{\bot\}$. *There exists a total computable function* $\mathrm{T}\colon \mathbb{N} \times \mathbb{S} \times \{L, R\}^* \to \mathbb{S}_\bot$, *such that for* $k \geq 1$

$$\mathrm{T}_0(\sigma, w) \;=\; \begin{cases} \sigma, & \text{if } w = \varepsilon, \\ \bot, & \text{otherwise,} \end{cases}$$

$$\mathrm{T}_k(\sigma, w) \;=\; \begin{cases} T_{k-1}(\tau, w'), & \text{if } \sigma = \langle P, \eta, a, \theta \rangle \vdash \langle P', \eta', a', \theta \cdot b \rangle = \tau, \\ & \quad \text{with } w = b \cdot w' \text{ and } b \in \{L, R, \varepsilon\}, \\ \bot & \text{otherwise.} \end{cases}$$

So $\mathrm{T}_k(\sigma, w)$ returns a successor state τ, if $\sigma \vdash^k \tau$, whereupon exactly $|w|$ inferences must use the (prob1)– or the (prob2)–rule and those probabilistic choices are resolved according to w. Otherwise $\mathrm{T}_k(\sigma, w)$ returns $\bot$. Note in particular that for both the inference of a terminal state $\langle \downarrow, \eta, a, \theta \rangle$ within less than k steps as well as the inference of a terminal state through less or more than $|w|$ probabilistic choices, the calculation of $\mathrm{T}_k(\sigma, w)$ will result in $\bot$. In addition to T, we will need two more computable operations for expressing expected outcomes, termination probabilities, and expected runtimes:

Proposition 2. *There exist two total computable functions* $\alpha\colon \mathbb{S}_\bot \to \mathbb{Q}^+$ *and* $\wp\colon \mathbb{S}_\bot \times \mathsf{Var} \to \mathbb{Q}^+$, *such that*

$$\alpha(\sigma) \;=\; \begin{cases} a, & \text{if } \sigma = \langle \downarrow, \_, a, \_ \rangle \\ 0, & \text{otherwise,} \end{cases} \qquad \wp(\sigma, v) \;=\; \begin{cases} \eta(v) \cdot a, & \text{if } \sigma = \langle \downarrow, \eta, a, \_ \rangle \\ 0, & \text{otherwise,} \end{cases}$$

where _ *represents an arbitrary value.*

The function α takes a state σ and returns the probability of reaching σ. The function $\wp$ takes a state σ and a variable v and returns the probability of reaching σ multiplied with the value of v in the state σ. Both functions do that only if the provided state σ is a terminal state. Otherwise they return 0. Based on the above notions, we now definie expected outcomes, termination probabilities and expected times until termination:

Definition 5 (Expected Outcome, Termination Probability, and Expected Time until Termination). *Let* $P \in \mathsf{Prog}$, $\eta \in \mathbb{V}$, $v \in \mathsf{Var}$, $\sigma_{P,\eta} = \langle P, \eta, 1, \varepsilon \rangle$, *and for a finite alphabet* A *let* $A^{\leq k} = \bigcup_{i=0}^{k} A^i$. *Then*

1. *the* ***expected outcome*** *of* v *after executing* P *on* η, *denoted* $\mathbf{E}_{P,\eta}(v)$, *is*

$$\mathrm{E}_{P,\eta}(v) \;=\; \sum_{k=0}^{\infty} \sum_{w \in \{L, R\}^{\leq k}} \wp\big(\mathrm{T}_k(\sigma_{P,\eta}, w), v\big) \;,$$

2. *the* ***probability that P terminates*** *on η, denoted* $\mathbf{Pr}_{P,\eta}(\downarrow)$*, is*

$$\Pr_{P,\eta}(\downarrow) \;=\; \sum_{k=0}^{\infty} \sum_{w \in \{L,\,R\}^{\leq k}} \alpha\big(\mathrm{T}_k(\sigma_{P,\eta},\, w)\big) \;,$$

3. *the* ***expected time until termination of P*** *on η, denoted* $\mathbf{E}_{P,\eta}(\downarrow)$*, is*

$$\mathrm{E}_{P,\eta}(\downarrow) \;=\; \sum_{k=0}^{\infty} \left(1 - \sum_{w \in \{L,\,R\}^{\leq k}} \alpha\big(\mathrm{T}_k(\sigma_{P,\eta},\, w)\big) \right) \;.$$

The expected outcome $\mathrm{E}_{P,\eta}(v)$ as defined here coincides with the weakest pre–expectation $wp.P.v\,(\eta)$ à la McIver and Morgan [4] for fully probabilistic programs. In the above definition for $\mathrm{E}_{P,\eta}(v)$, we sum over all possible numbers of inference steps k and sum over all possible sequences from length 0 up to length k for resolving all probabilistic choices. Using $\wp$ we filter out the terminal states σ and sum up the values of $\wp(\sigma,\, v)$.

For the termination probability $\Pr_P(\downarrow)$, we basically do the same but we merely sum up the probabilities of reaching final states by using α instead of $\wp$.

For the expected time until termination $\mathrm{E}_{P,\eta}(\downarrow)$, we go along the lines of [6]: It is stated there that the expected time until termination of P on η can be expressed as $\sum_{k=0}^{\infty} \Pr(\text{“}P \text{ runs for more than } k \text{ steps on } \eta\text{”}) = \sum_{k=0}^{\infty} \big(1 - \Pr(\text{“}P \text{ terminates within } k \text{ steps on } \eta\text{”})\big)$. We have expressed the latter in our set–up.

In order to investigate the complexity of calculating $\mathrm{E}_{P,\eta}(v)$, we define three sets: $\mathcal{LEXP}$, which relates to the set of rational lower bounds of $\mathrm{E}_{P,\eta}(v)$, $\mathcal{REXP}$, which relates to the set of rational upper bounds, and $\mathcal{EXP}$ which relates to the value of $\mathrm{E}_{P,\eta}(v)$ itself:

Definition 6 ($\mathcal{LEXP}$, $\mathcal{REXP}$, and $\mathcal{EXP}$). *The sets $\mathcal{LEXP}, \mathcal{REXP}, \mathcal{EXP} \subset \mathsf{Prog} \times \mathbb{V} \times \mathsf{Var} \times \mathbb{Q}^+$ are defined as $(P,\, \eta,\, v,\, q) \in \mathcal{LEXP}$ iff $q < \mathrm{E}_{P,\eta}(v)$, $(P,\, \eta,\, v,\, q) \in \mathcal{REXP}$ iff $q > \mathrm{E}_{P,\eta}(v)$, and $(P,\, \eta,\, v,\, q) \in \mathcal{EXP}$ iff $q = \mathrm{E}_{P,\eta}(v)$.*

Regarding the termination probability of a probabilistic program, the case of almost–sure termination is of special interest: We say that a program P *terminates almost–surely* on input η iff P terminates on η with probability 1. Furthermore, we say that P *terminates positively almost–surely* on η iff the expected time until termination of P on η is finite. Lastly, we say that P terminates *universally* (positively) almost–surely, if it does so on all possible inputs η. The problem of (universal) almost–sure termination can be seen as the probabilistic counterpart to the (universal) halting problem for ordinary programs.

In the following, we formally define the according problem sets:

Definition 7 (Almost–Sure Termination Problem Sets). *The sets $\mathcal{AST}$, $\mathcal{PAST}$, $\mathcal{UAST}$, and $\mathcal{PAST}$ are defined as follows:*

$$(P,\, \eta) \in \mathcal{AST} \iff \Pr_{P,\eta}(\downarrow) = 1 \qquad (P,\, \eta) \in \mathcal{PAST} \iff \mathrm{E}_{P,\eta}(\downarrow) < \infty$$
$$P \in \mathcal{UAST} \iff \forall \eta\colon (P,\, \eta) \in \mathcal{AST} \qquad P \in \mathcal{UPAST} \iff \forall \eta\colon (P,\, \eta) \in \mathcal{PAST}$$

Notice that both $\mathcal{PAST} \subset \mathcal{AST}$ and $\mathcal{UPAST} \subset \mathcal{UAST}$ hold.

5 The Hardness of Computing Expected Outcomes

In this section we investigate the computational hardness of deciding the sets $\mathcal{LEXP}$, $\mathcal{REXP}$, and $\mathcal{EXP}$. The first fact we establish is the Σ_1^0–completeness of $\mathcal{LEXP}$. This result is established by reduction from the (non–universal) halting problem for ordinary programs:

Theorem 1 (The Halting Problem [16]). *The* ***halting problem*** *is a subset* $\boldsymbol{\mathcal{H}} \subset \mathsf{ordProg} \times \mathbb{V}$*, which is characterized as* $(P, \eta) \in \mathcal{H}$ *iff* $\exists k\, \exists \eta' \colon \mathrm{T}_k(\sigma_{P,\eta}, \varepsilon) = \langle \downarrow, \eta', 1, \varepsilon \rangle$*. Let* $\overline{\boldsymbol{\mathcal{H}}}$ *denote the* ***complement of the halting problem****, i.e.* $\overline{\mathcal{H}} = (\mathsf{ordProg} \times \mathbb{V}) \setminus \mathcal{H}$*.* $\mathcal{H}$ *is* Σ_1^0*–complete and* $\overline{\mathcal{H}}$ *is* Π_1^0*–complete.*

Theorem 2. $\mathcal{LEXP}$ *is* Σ_1^0*–complete.*

Proof. For $\mathcal{LEXP} \in \Sigma_1^0$ consider the following:

$$
\begin{aligned}
& (P, \eta, v, q) \in \mathcal{LEXP} \\
\iff\ & q < \mathrm{E}_{P,\eta}(v) \\
\iff\ & q < \sum_{k=0}^{\infty} \sum_{w \in \{L,R\}^{\leq k}} \wp\big(\mathrm{T}_k(\sigma_{P,\eta}, w), v\big) \\
\iff\ & \exists\, y \colon q < \sum_{k=0}^{y} \sum_{w \in \{L,R\}^{\leq k}} \wp\big(\mathrm{T}_k(\sigma_{P,\eta}, w), v\big) \\
\implies\ & \mathcal{LEXP} \in \Sigma_1^0
\end{aligned}
$$

Figure 1 (left) gives an intuition on this formula.

It remains to show that $\mathcal{LEXP}$ is Σ_1^0–hard: We do this by proving $\mathcal{H} \leq_{\mathrm{m}} \mathcal{LEXP}$. Consider the following function $f \colon \mathcal{H} \leq_{\mathrm{m}} \mathcal{LEXP}$: f takes an ordinary program $Q \in \mathsf{ordProg}$ and a variable valuation η as its input and computes $(P, \eta, v, 1/2)$, where P is the probabilistic program $v := 0; \{v := 1\}[1/2]\{TQ; v := 1\}$ and TQ is an ordinary program that simulates Q on η.

Correctness of the reduction: There are two cases: (1) Q terminates on input η. Then the expected outcome of variable v after executing the program P on input η is 1, because in both branches, the variable will be set to 1. As $1/2 < 1$, we have that $(P, \eta, v, 1/2) \in \mathcal{LEXP}$.

(2) Q does not terminate on input η. Then the expected outcome of variable v after executing the program P on input η is $1/2$, because only in the left branch (which has probability $1/2$), the variable will be set to 1. In the right branch, the program does not terminate and therefore the outcome of this branch is 0. As $1/2 \not< 1/2$, we have that $(P, \eta, v, 1/2) \notin \mathcal{LEXP}$. □

Theorem 2 implies that $\mathcal{LEXP}$ is recursively enumerable. This means that all lower bounds for expected outcomes can be effectively enumerated by some algorithm. Now, if upper bounds were recursively enumerable as well, then expected outcomes would be computable reals. However, the contrary will be shown by establishing that $\mathcal{REXP}$ is Σ_2^0–complete, thus $\mathcal{REXP} \notin \Sigma_1^0$ and hence $\mathcal{REXP}$

is not recursively enumerable. Σ_2^0–hardness will be established by a reduction from the complement of the universal halting problem for ordinary programs:

Theorem 3 (The Universal Halting Problem [16]). *The* ***universal halting problem*** *is a subset* $\boldsymbol{\mathcal{UH}} \subset \mathsf{ordProg}$*, which is characterized as* $P \in \mathcal{UH}$ *iff* $\forall\, \eta\colon (P, \eta) \in \mathcal{H}$*. Let* $\overline{\boldsymbol{\mathcal{UH}}}$ *denote the* ***complement of*** $\boldsymbol{\mathcal{UH}}$*, i.e.,* $\overline{\mathcal{UH}} = \mathsf{ordProg} \setminus \mathcal{UH}$*.* $\mathcal{UH}$ *is* Π_2^0*–complete and* $\overline{\mathcal{UH}}$ *is* Σ_2^0*–complete.*

Theorem 4. $\mathcal{REXP}$ *is* Σ_2^0*–complete.*

Proof. For $\mathcal{REXP} \in \Sigma_2^0$ consider the following:

$$\begin{aligned}
& (P, \eta, v, q) \in \mathcal{REXP} \\
\iff\ & q > \mathrm{E}_{P,\eta}(v) \\
\iff\ & q > \sum_{k=0}^{\infty} \sum_{w \in \{L,R\}^{\leq k}} \wp\big(\mathrm{T}_k(\sigma_{P,\eta}, w), v\big) \\
\iff\ & \exists\, \delta\, \forall\, y\colon q - \delta > \sum_{k=0}^{y} \sum_{w \in \{L,R\}^{\leq k}} \wp\big(\mathrm{T}_k(\sigma_{P,\eta}, w), v\big) \\
\implies\ & \mathcal{REXP} \in \Sigma_2^0
\end{aligned}$$

Figure 1 (right) gives an intuition on this formula.

It remains to show that $\mathcal{REXP}$ is Σ_2^0–hard: We do this by proving $\overline{\mathcal{UH}} \leq_{\mathrm{m}} \mathcal{REXP}$. Consider the following function $f\colon \overline{\mathcal{UH}} \leq_{\mathrm{m}} \mathcal{REXP}$: f takes an ordinary program $Q \in \mathsf{ordProg}$ as its input and computes the triple $(P, \eta, v, 1)$, where η is an arbitrary but fixed input, and $P \in \mathsf{Prog}$ is the following probabilistic program:

```
i := 0;  {c := 0} [0.5] {c := 1};
while (c ≠ 0){ i := i + 1; {c := 0} [0.5] {c := 1} };

k := 0; {c := 0} [0.5] {c := 1};
while (c ≠ 0){ k := k + 1; {c := 0} [0.5] {c := 1} };

v := 0; TQ
```

TQ is a program that computes $\alpha\big(\mathrm{T}_k\big(\langle Q, g_Q(i), 1, \varepsilon\rangle, \varepsilon\big)\big) \cdot 2^{k+1}$ and stores the result in the variable v, and $g_Q\colon \mathbb{N} \to \mathbb{V}$ is some computable bijection, such that $\forall z \in \mathsf{Var}\colon \big(g_Q(i)\big)(z) \neq 0$ implies that z occurs in Q.

Correctness of the reduction: $\alpha\big(\mathrm{T}_k\big(\langle Q, g_Q(i), 1, \varepsilon\rangle, \varepsilon\big)\big) \cdot 2^{k+1}$ returns 2^{k+1} if and only if Q halts on input $g_Q(i)$ after exactly k steps (otherwise it returns 0). The two while–loops generate independent geometric distributions with parameter $1/2$ on i and k, respectively, so the probability of generating exactly the numbers i and k is $1/2^{k+1} \cdot 1/2^{k+1} = 1/2^{i+k+2}$. The expected outcome of v after executing the program P on any input η is hence

$$\sum_{i=0}^{\infty} \sum_{k=0}^{\infty} \frac{1}{2^{i+k+2}} \cdot \alpha\Big(\mathrm{T}_k\big(\langle Q, g_Q(i), 1, \varepsilon\rangle, \varepsilon\big)\Big) \cdot 2^{k+1}\ .$$

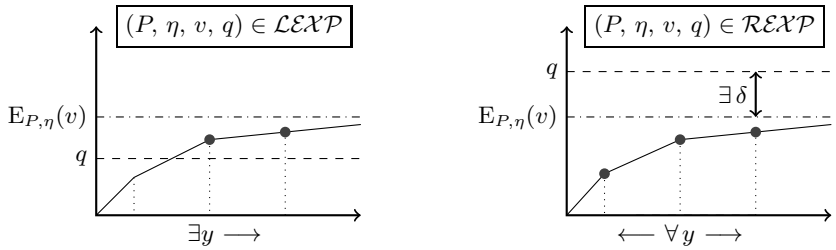

Fig. 1. Schematic depiction of the formulae defining $\mathcal{LEXP}$ and $\mathcal{REXP}$, respectively. In each diagram, the solid line represents the monotonically increasing graph of $\sum_{k=0}^{y} \sum_{w \in \{L,\, R\}^{\leq k}} \wp\big(\mathrm{T}_k(\sigma_{P,\eta},\, w),\, v\big)$ plotted over increasing y.

Since for each input, the number of steps until termination of Q is either unique or does not exist, the formula for the expected outcome reduces to $\sum_{i=0}^{\infty} \frac{1}{2^{i+1}} = 1$ if and only if Q halts on every input after some finite number of steps. Thus if there exists an input on which Q *does not* eventually halt, then $(P,\, \eta,\, v,\, 1) \in \mathcal{REXP}$ as then the expected value is strictly less than one. If, on the other hand, Q *does* halt on every input, then the expected outcome is exactly one and hence $(P,\, \eta,\, v,\, 1) \notin \mathcal{REXP}$. □

Finally, we establish the following result regarding exact expected outcomes:

Theorem 5. *$\mathcal{EXP}$ is Π_2^0–complete.*

Proof. For $\mathcal{EXP} \in \Pi_2^0$ consider the following: By Theorem 4, there exists a decidable relation $\mathcal{R}$, such that $(P,\, \eta,\, v,\, x) \in \mathcal{REXP}$ iff $\exists r_1 \forall r_2 \colon (r_1,\, r_2,\, P,\, \eta,\, v,\, x) \in \mathcal{R}$. Furthermore from Theorem 2 it follows that there exists a decidable relation $\mathcal{L}$, such that $(P,\, \eta,\, v,\, x) \in \mathcal{LEXP}$ iff $\exists \ell \colon (\ell,\, P,\, \eta,\, v,\, x) \in \mathcal{L}$. Let $\neg\mathcal{R}$ and $\neg\mathcal{L}$ be the (decidable) negations of $\mathcal{R}$ and $\mathcal{L}$, respectively, then:

$$
\begin{aligned}
& (P,\, \eta,\, v,\, q) \in \mathcal{EXP} \\
\iff\ & q = \mathrm{E}_{P,\eta}(v) \\
\iff\ & q \leq \mathrm{E}_{P,\eta}(v) \,\wedge\, q \geq \mathrm{E}_{P,\eta}(v) \\
\iff\ & \neg\big(q > \mathrm{E}_{P,\eta}(v)\big) \,\wedge\, \neg\big(q < \mathrm{E}_{P,\eta}(v)\big) \\
\iff\ & \neg\big(\exists r_1\, \forall r_2 \colon (r_1,\, r_2,\, P,\, \eta,\, v,\, q) \in \mathcal{R}\big) \,\wedge\, \neg\big(\exists \ell \colon (\ell,\, P,\, \eta,\, v,\, q) \in \mathcal{L}\big) \\
\iff\ & \big(\forall r_1\, \exists r_2 \colon (r_1,\, r_2,\, P,\, \eta,\, v,\, q) \in \neg\mathcal{R}\big) \,\wedge\, \big(\forall \ell \colon (\ell,\, P,\, \eta,\, v,\, q) \in \neg\mathcal{L}\big) \\
\iff\ & \forall r_1\, \forall \ell\, \exists r_2 \colon (r_1,\, r_2,\, P,\, \eta,\, v,\, q) \in \neg\mathcal{R} \,\wedge\, (\ell,\, P,\, \eta,\, v,\, q) \in \neg\mathcal{L} \\
\implies\ & \mathcal{EXP} \in \Pi_2^0
\end{aligned}
$$

It remains to show that $\mathcal{EXP}$ is Π_2^0–hard. We do this by proving $\mathcal{UH} \leq_{\mathrm{m}} \mathcal{EXP}$. Consider again the reduction function f from the proof of Theorem 4: Given an ordinary program Q, f computes the triple $(P,\, \eta,\, v,\, 1)$, where P is a probabilistic program which has an expected outcome of one for the variable v if and only if Q terminates on all inputs, which is nothing else than $Q \in \mathcal{UH}$. Thus $f \colon \mathcal{UH} \leq_{\mathrm{m}} \mathcal{EXP}$. □

6 The Hardness of Deciding Probabilistic Termination

This section presents the main contributions of this paper: Hardness results on several variations of almost–sure termination problems. We first establish that deciding almost–sure termination of a program on a given input is Π^0_2–complete:

Theorem 6. *$\mathcal{AST}$ is Π^0_2–complete.*

Proof. For $\mathcal{AST} \in \Pi^0_2$ we show $\mathcal{AST} \leq_{\mathrm{m}} \mathcal{EXP}$. For that, consider the following function $f\colon \mathcal{AST} \leq_{\mathrm{m}} \mathcal{EXP}$ which takes a probabilistic program Q and an input η as its input and computes the tuple $(P, \eta, v, 1)$, where $v \in \mathsf{Var}$ does not occur in Q and P is the probabilistic program $v := 0;\ Q;\ v := 1$

Correctness of the reduction: On executing P, the variable v is set to one only in those runs in which the program Q terminates on η. So the expected value of v converges to one, if and only if the probability of Q terminating converges to one. So if $(Q, \eta) \in \mathcal{AST}$, then and only then $(P, \eta, v, 1) \in \mathcal{EXP}$. Thus $f\colon \mathcal{AST} \leq_{\mathrm{m}} \mathcal{EXP}$. By Theorem 5, $\mathcal{EXP}$ is Π^0_2–complete, so it follows directly that $\mathcal{AST} \in \Pi^0_2$.

It remains to show that $\mathcal{AST}$ is Π^0_2–hard. For that we many–one reduce the Π^0_2–complete universal halting problem to $\mathcal{AST}$ using the following function $f'\colon \mathcal{UH} \leq_{\mathrm{m}} \mathcal{AST}$: f' takes an ordinary program Q as its input and computes the pair (P', η), where η is some arbitrary but fixed input and P' is the following probabilistic program:

```
i := 0; {c := 0} [0.5] {c := 1};
while (c ≠ 0){ i := i + 1; {c := 0} [0.5] {c := 1} };
SQ
```

where SQ is an ordinary program that on any input η simulates the program Q on input $g_Q(i)$, and $g_Q\colon \mathbb{N} \to \mathbb{V}$ is some computable bijection, such that $\forall v \in \mathsf{Var}\colon \big(g_Q(i)\big)(v) \neq 0$ implies that v occurs in Q.

Correctness of the reduction: The while–loop in P' establishes a geometric distribution with parameter $1/2$ on i and hence a geometric distribution on all possible inputs for Q. After the while–loop, the program Q is simulated on the input generated probabilistically in the while–loop. Obviously then the entire program P' terminates with probability one on any arbitrary input η, i.e. terminates almost–surely on η, if and only if the simulation of Q terminates on every input. Thus $Q \in \mathcal{UH}$ if and only if $(P', \eta) \in \mathcal{AST}$. □

While for ordinary programs there is a complexity leap when moving from the halting problem for some given input to the universal halting problem, we establish that *there is no such leap in the probabilistic setting*, i.e. $\mathcal{UAST}$ is as hard as $\mathcal{AST}$:

Theorem 7. *$\mathcal{UAST}$ is Π^0_2–complete.*

Proof. For showing $\mathcal{UAST} \in \Pi^0_2$, consider that by Theorem 6 there must exist a decidable relation $\mathcal{R}$ such that $(P, \eta) \in \mathcal{AST}$ iff $\forall\, y_1\, \exists\, y_2\colon (y_1, y_2, P, \eta) \in \mathcal{R}$.

By that we have that $P \in \mathcal{UAST}$ iff $\forall\, \eta\, \forall\, y_1\, \exists\, y_2 \colon (y_1, y_2, P, \eta) \in \mathcal{R}$, which is a Π_2^0–formula. It remains to show that $\mathcal{UAST}$ is Π_2^0–hard. This can be done by proving $\mathcal{AST} \leq_{\mathrm{m}} \mathcal{UAST}$ as follows: On input (Q, η) the reduction function $f \colon \mathcal{AST} \leq_{\mathrm{m}} \mathcal{UAST}$ computes a probabilistic program P that first initializes all variables according to η and then executes Q. □

We now investigate the computational hardness of deciding *positive* almost–sure termination: It turns out that deciding $\mathcal{PAST}$ is—although still undecidable—computationally more benign than deciding $\mathcal{AST}$, namely it is Σ_2^0–complete. Thus $\mathcal{PAST}$ becomes semi–decidable when given access to an $\mathcal{H}$–oracle whereas $\mathcal{AST}$ does not. We establish Σ_2^0–hardness by a reduction from $\overline{\mathcal{UH}}$. This result is particularly counterintuitive as it means that for each ordininary program that *does not halt* on all inputs, we can *compute* a probabilistic program that *does halt* within an expected finite number of steps.

Theorem 8. *$\mathcal{PAST}$ is Σ_2^0–complete.*

Proof. For $\mathcal{PAST} \in \Sigma_2^0$ consider the following:

$$
\begin{aligned}
& (P, \eta) \in \mathcal{PAST} \\
\iff\ & \infty > \mathrm{E}_{P,\eta}(\downarrow) \\
\iff\ & \exists\, c \colon c > \mathrm{E}_{P,\eta}(\downarrow) \\
\iff\ & \exists\, c \colon c > \sum_{k=0}^{\infty} \left(1 - \sum_{w \in \{L,\, R\}^{\leq k}} \alpha\big(\mathrm{T}_k(\sigma_{P,\eta},\, w)\big) \right) \\
\iff\ & \exists\, c\, \forall\, \ell \colon c > \sum_{k=0}^{\ell} \left(1 - \sum_{w \in \{L,\, R\}^{\leq k}} \alpha\big(\mathrm{T}_k(\sigma_{P,\eta},\, w)\big) \right) \\
\implies\ & \mathcal{PAST} \in \Sigma_2^0
\end{aligned}
$$

It remains to show that $\mathcal{PAST}$ is Σ_2^0–hard. For that we use a reduction function $f \colon \overline{\mathcal{UH}} \leq_{\mathrm{m}} \mathcal{PAST}$ with $f(Q) = (P, \eta)$, where η is arbitrary but fixed and P is the program

```
c := 1; i := 0; x := 0; term := 0; InitQ;

while (c ≠ 0){
    StepQ;
    if (term = 1){
        Cheer; i := i + 1; term := 0; InitQ
    };
    {c := 0} [0.5] {c := 1}; x := x + 1
} ,
```

where $\mathit{InitQ} \in \mathsf{ordProg}$ is a program that initializes a simulation of the program Q on input $g_Q(i)$ (recall the bijection $g_Q \colon \mathbb{N} \to \mathbb{V}$ from Theorem 4), $\mathit{StepQ} \in \mathsf{ordProg}$ is a program that does one single (further) step of that simulation and

sets `term` to 1 if that step has led to termination of Q, and *Cheer* $\in$ ordProg is a program that executes 2^x many effectless steps. In the following we refer to this as "cheering"[2].

Correctness of the reduction: Intuitively, the program P starts by simulating Q on input $g_Q(0)$. During the simulation, it—figuratively speaking—gradually looses interest in further simulating Q by tossing a coin after each simulation step to decide whether to continue the simulation or not. If eventually P finds that Q has halted on input $g_Q(0)$, it "cheers" for a number of steps exponential in the number of coin tosses that were made so far, namely for 2^x steps. P then continues with the same procedure for the next input $g_Q(1)$, and so on.

The variable x keeps track of the number of loop iterations (starting from 0), which equals the number of coin tosses. The x–th loop iteration takes place with probability $1/2^x$. One loop iteration consists of a constant number of steps c_1 in case Q did not halt on input $g_Q(i)$ in the current simulation step. Such an iteration therefore contributes $c_1/2^x$ to the expected runtime of the probabilistic program P. In case Q did halt, a loop iteration takes a constant number of steps c_2 plus 2^x additional "cheering" steps. Such an iteration therefore contributes $c_2+2^x/2^x = c_2/2^x + 1 > 1$ to the expected runtime. Overall, the expected runtime of the program P roughly resembles a geometric series with exponentially decreasing summands. However, for each time the program Q halts on an input, a summand of the form $c_2/2^x + 1$ appears in this series. There are now two cases:

(1) $Q \in \overline{\mathcal{UH}}$, so there exists some input η with minimal i such that $g_Q(i) = \eta$ on which Q does not terminate. In that case, summands of the form $c_2/2^x + 1$ appear only $i - 1$ times in the series and therefore, the series converges—the expected time until termination is finite, so $(P, \eta) \in \mathcal{PAST}$.

(2) $Q \notin \overline{\mathcal{UH}}$, so Q terminates on every input. In that case, summands of the form $c_2/2^x + 1$ appear infinitely often in the series and therefore, the series diverges—the expected time until termination is infinite, so $(P, \eta) \notin \mathcal{PAST}$. □

The final problem we study is *universal* positive almost–sure termination. In contrast to the non–positive version, we do have a complexity leap when moving from non–universal to universal positive almost–sure termination. We will establish that $\mathcal{UPAST}$ is Π^0_3–complete and thus even harder to decide than $\mathcal{UAST}$. For the reduction, we make use of the following Π^0_3–complete problem:

Theorem 9 (The Cofiniteness Problem [16]). *The* ***cofiniteness problem*** *is a subset* $\boldsymbol{\mathcal{COF}} \subset$ ordProg*, which is characterized as* $P \in \mathcal{COF}$ *iff* $\{\eta \mid (P, \eta) \in \mathcal{H}\}$ *is cofinite. Let* $\overline{\boldsymbol{\mathcal{COF}}}$ *denote the* ***complement of*** $\boldsymbol{\mathcal{COF}}$*, i.e.* $\overline{\mathcal{COF}} =$ ordProg$\setminus \mathcal{COF}$*.* $\mathcal{COF}$ *is* Σ^0_3*–complete and* $\overline{\mathcal{COF}}$ *is* Π^0_3*–complete.*

Theorem 10. $\mathcal{UPAST}$ *is* Π^0_3*–complete.*

Proof. By Theorem 8, there exists a decidable relation $\mathcal{R}$, such that $(P, \eta) \in \mathcal{PAST}$ iff $\exists\, y_1\, \forall\, y_2 \colon (y_1, y_2, P, \eta) \in \mathcal{R}$. Therefore $\mathcal{UPAST}$ is definable by $P \in \mathcal{UPAST}$ iff $\forall\, \eta\, \exists\, y_1\, \forall\, y_2 \colon (y_1, y_2, P, \eta) \in \mathcal{R}$ which gives a Π^0_3–formula.

[2] The program P cheers as it was able to prove the termantion of Q on input $g_Q(i)$.

It remains to show that $\mathcal{UPAST}$ is Π^0_3–hard. For that we many–one reduce the Π^0_3–complete complement of the cofiniteness problem to $\mathcal{UPAST}$ using the following function $f\colon \overline{\mathcal{COF}} \leq_{\mathrm{m}} \mathcal{UPAST}$: f takes an ordinary program Q as its input and computes the following probabilistic program P:

```
c := 1; x := 0; term := 0; InitQ;
while (c ≠ 0){
    StepQ; if (term = 1){Cheer; i := i + 1; term := 0; InitQ}
    x := x + 1; {c := 0} [0.5] {c := 1}
},
```

where *InitQ* is a program that initializes a simulation of the program Q on input $g_Q(i)$ (recall the bijection $g_Q\colon \mathbb{N} \to \mathbb{V}$ from Theorem 4), *StepQ* is a program that does one single (further) step of that simulation and sets `term` to 1 if that step has led to termination of Q, and *Cheer* is a program that executes 2^x many effectless steps.

Correctness of the reduction: The only difference to the program from the proof of Theorem 8 is that the variable i is not initialized with 0. Thus, on input η the program P also simulates Q succesively on all inputs but starting from input $g_Q\big(\eta(i)\big)$ instead of $g_Q(0)$.

The problem $\overline{\mathcal{COF}}$ can alternatively be defined as $Q \in \overline{\mathcal{COF}}$ iff $\big\{\eta \mid (Q, \eta) \in \overline{\mathcal{H}}\big\}$ is infinite. There are now two cases:

(1) $Q \in \overline{\mathcal{COF}}$. Thus, there are infinitely many inputs on which Q does not terminate. So no matter with which number $\eta(i) \in \mathbb{N}$ the variable i is initialized, the variable i will eventually be incremented to some value j such that Q does not terminate on $g_Q(j)$ and therefore, in the while–loop the `if`–branch with the "cheering" steps will eventually *not* be executed anymore. Consequently, the expected time until termination of P on any input η is finite and therefore $P \in \mathcal{UPAST}$.

(2) $Q \notin \overline{\mathcal{COF}}$. Then there are only finitely many inputs on which Q does not terminate. Say j is minimal such that Q does not terminate on input $g_Q(j)$, i.e. the program Q terminates on all other inputs $g_Q(i')$ with $j < i'$. So when running the program P on some input η with $\eta(i) > j$, in the while–loop the `if`–branch with the "cheering" steps will be executed infinitely often. Consequently, the expected time until termination of P on that input η is infinite and therefore $P \notin \mathcal{UPAST}$. □

7 Conclusion

We have studied the computational complexity of solving a variety of natural problems which appear in the analysis of probabilistic programs: Computing lower bounds, upper bounds, and exact expected outcomes ($\mathcal{LEXP}$, $\mathcal{REXP}$, and $\mathcal{EXP}$), deciding non–universal and universal almost–sure termination ($\mathcal{AST}$ and $\mathcal{UAST}$), and deciding non–universal and universal positive almost–sure termination ($\mathcal{PAST}$ and $\mathcal{UPAST}$). Our complexity results are summarized in Figure 2.

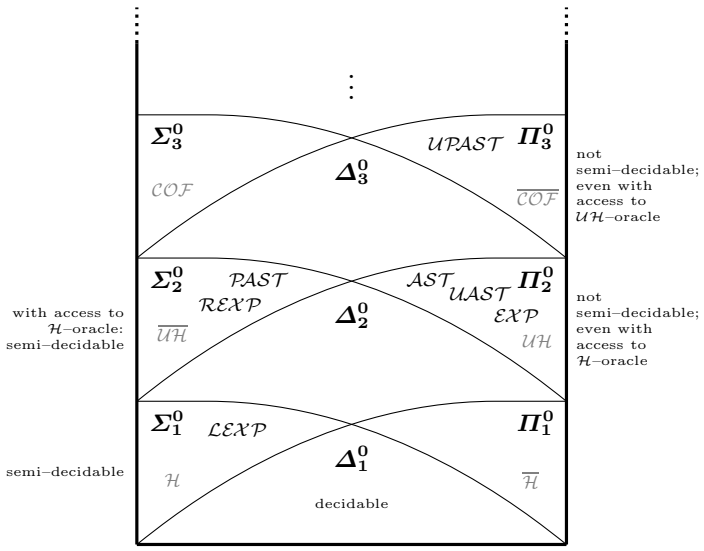

Fig. 2. The complexity landscape of determining expected outcomes and deciding (universal) (positive) almost–sure termination.

All examined problems are complete for their respective level of the arithmetical hierarchy.

Future work consists of identifying program subclasses for which some of the studied problems become easier. One idea towards this would be to investigate the use of quantifier–elimination methods such as e.g. Skolemization.

Acknowledgements The authors would like to thank Luis María Ferrer Fioriti (Saarland University) and Federico Olmedo (RWTH Aachen) for the fruitful discussions on the topics of this paper.

References

1. Kozen, D.: Semantics of Probabilistic Programs. J. Comput. Syst. Sci. **22**(3) (1981) 328–350
2. Barthe, G., Köpf, B., Olmedo, F., Béguelin, S.Z.: Probabilistic Relational Reasoning for Differential Privacy. ACM Trans. Program. Lang. Syst. **35**(3) (2013) 9
3. Borgström, J., Gordon, A., Greenberg, M., Margetson, J., van Gael, J.: Measure Transformer Semantics for Bayesian Machine Learning. LMCS **9**(3) (2013)
4. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Springer (2004)
5. Gretz, F., Katoen, J.P., McIver, A.: Operational versus Weakest Pre–Expectation Semantics for the Probabilistic Guarded Command Language. Performance Evaluation **73** (2014) 110–132

6. Fioriti, L.M.F., Hermanns, H.: Probabilistic termination: Soundness, completeness, and compositionality. In: POPL 2015, ACM (2015) 489–501
7. Sneyers, J., de Schreye, D.: Probabilistic Termination of CHRiSM Programs. In: LOPSTR. Volume 7225 of LNCS., Springer (2011) 221–236
8. Arons, T., Pnueli, A., Zuck, L.D.: Parameterized Verification by Probabilistic Abstraction. In: FoSSaCS. Volume 2620 of LNCS., Springer (2003) 87–102
9. Esparza, J., Gaiser, A., Kiefer, S.: Proving Termination of Probabilistic Programs Using Patterns. In: CAV. Volume 7358 of LNCS., Springer (2012) 123–138
10. Morgan, C.: Proof Rules for Probabilistic Loops. In: Proc. of the BCS-FACS 7th Refinement Workshop, Workshops in Computing, Springer Verlag (1996) 7
11. Tiomkin, M.L.: Probabilistic Termination Versus Fair Termination. TCS **66**(3) (1989) 333–340
12. Kleene, S.C.: Recursive Predicates and Quantifiers. Trans. of the AMS **53**(1) (1943) 41 – 73
13. Odifreddi, P.: Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers. Elsevier (1992)
14. Post, E.L.: Recursively Enumerable Sets of Positive Integers and their Decision Problems. Bulletin of the AMS **50**(5) (1944) 284–316
15. Davis, M.D.: Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science. Academic Press (1994)
16. Odifreddi, P.: Classical Recursion Theory, Volume II. Elsevier (1999)